Cognitive class ai
Spark fundamentals I

M T W T F S S
Page No.:
Date:
YOUVA

* Big data and spark
  - Data is increasing in volume, velocity, variety
  - The need to have faster results from analytics becomes increasingly important
  - Apache spark is a computing platform designed to be fast and general-purpose and easy to use

Speed
- In memory computations
- Faster than MapReduce for complex applications on disk

Generality:
- Covers a wide range of workloads on one system
- Batch applications (e.g. MapReduce)
- Iterative algorithms
- Interactive queries and streaming

- Ease of use
- API for Scala python Java
- Libraries for SQL, ML, streaming and graph processing
- Runs on Hadoop clusters or as a standalone

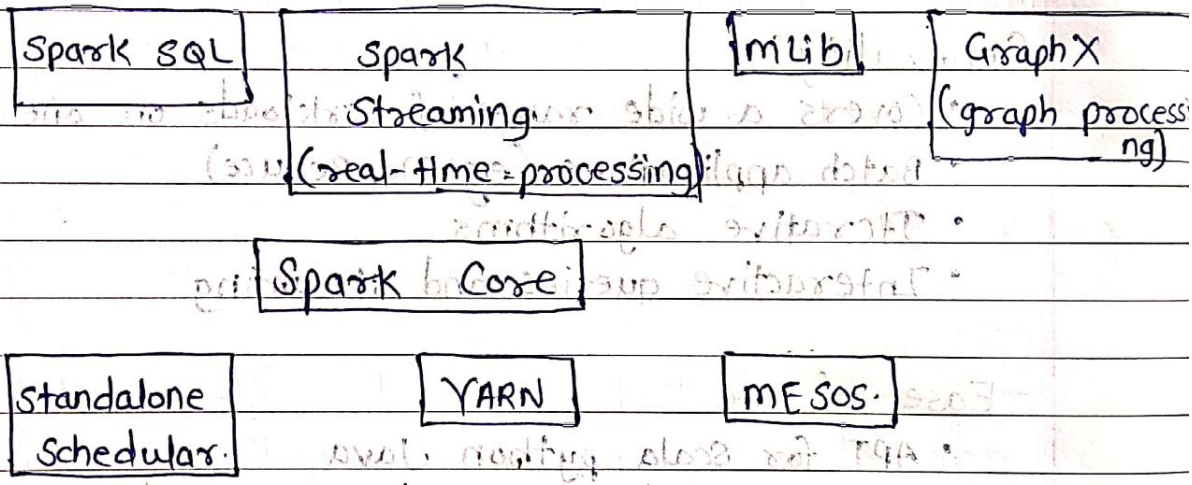
* Who uses spark and why?
- Parallel distributed processing, fault tolerance on comodity hardware, scalability, in memory computing, high level API's etc.
- Saves time and money

Data scientist
- Analyze and model the data to obtain insight during ad-hoc analysis
- Transforming the data into a useable format
- Statistics, machine learning, SQL

Engineers
- Develop a data processing systems or application
- Inspect and tune their application
- Programming with the spark's API

Everyone else
- Ease of use
- wide variety of functionality
- Mature and reliable

Spark unified stack



- Mapreduce started a general batch processing paradig
- Two limitations:-
  1) Difficulty programming in MapReduce
  2) Batch processing did not fit many use cases
- spawned a lot of specialized systems (storm, Impala, Giraph, etc)

Resilient Distributed Datasets (RDD)
- Spark primary abstraction
- Distribution collection of elements
- Parallelized across the cluster

- Two types of RDD operations
  - Transformations
    - Creates a DAG
    - Lazy evaluations
    - No return value
  - Actions
    - Performs the transformation and the action that follows
    - Return a value
- fault tolerance
- caching
- Example of RDD flow

Hadoop RDD → filtered RDD → mapped RDD → Reduced RDD

* Spark Jobs and shell
- spark Jobs can be written in scala, python, or Java
- spark shells for scala and python.
- APT's are available for all three
- must adhere to the appropriate versions for each spark release
- spark's native language is scala. So it is natural to write spark applications using ~~Java~~ scala.
- The course will cover code examples from scala, python and Java.

* Brief Overview of scala:
- Everything is an object
  - primitive types such as numbers or boolean
  - functions
- Numbers are objects
  - 1+2 * 3/4 → (1) + (((2) * (3)) / (x))

- where the +, -, *, / are valid identifiers in scala

- Functions are object:-
  - Pass functions as arguments
  - store them in variables
  - Return them from other functions
- function declaration
  - def functionName ([list of parameters]): [return type]

* Scala - anonymous functions
- functions without a name created for one-time use to pass to another function.
- Left side of the right arrow => is where the argument resides to (no arguments in eg)
- Right side of the arrow is the body of the function (the println statement)

Spark's scala and Python shell
- Spark's shell provides a simple way to learn the API
- Powerful tool to analyze data interactively
- The Scala shell runs on the Java VM
  - A good way to use existing Java libraries
- Scala:
  - To launch the Scala shell:
    ./bin/spark-shell
  - To read in text file:
    scala> val textfile = sc.textFile("Readme.md")

- Python:
  - To launch the python shell:
    ./bin/pyspark
  - To read in a text file

M T W T F S S
Page No.:
Date:
YOUVA

```
>>> textfile = sc.textFile("Readme.md").
```

Getting started with !spark - Part-1

Part-II

coding in Jupyter notebook

Resilient Distributed Dataset - Part 1

- fault tolerant of collection of elements that can be operated on in parallel.
- immutable
- Three methods for creating RDD
  - Parallelizing an existing collection.
  - Referencing a dataset
  - Transformation from an existing RDD

- Two types of RDD operations
  - Transformations
  - Actions

- Dataset from any storage supported by Hadoop
  - HDFS
  - cassandra
  - Hbase
  - Amazon S3
  - etc

- Types of file supported
  - text files
  - Sequence files
  - Hadoop input format

Creating an RDD

- Launch the Spark shell
  /bin/spark-shell
- Create some data
  val data = 1 to 10000
- Parallelize that data (creating the RDD)
  val distData = sc.parallelize(data)
- Perform additional transformations or invoke an action on it
  distData.filter(...)
- Alternatively, create an RDD from an external dataset
  - val readmeFile = sc.textFile("Readme.md").

RDD operations - Basics

- Loading a file
  val lines = sc.textFile("hdfs://data.txt")

- Applying transformation
  val linelengths = lines.map(s => s.length)

- Invoking action
  val totalLengths = lineLengths.reduces((a,b) => a+b)

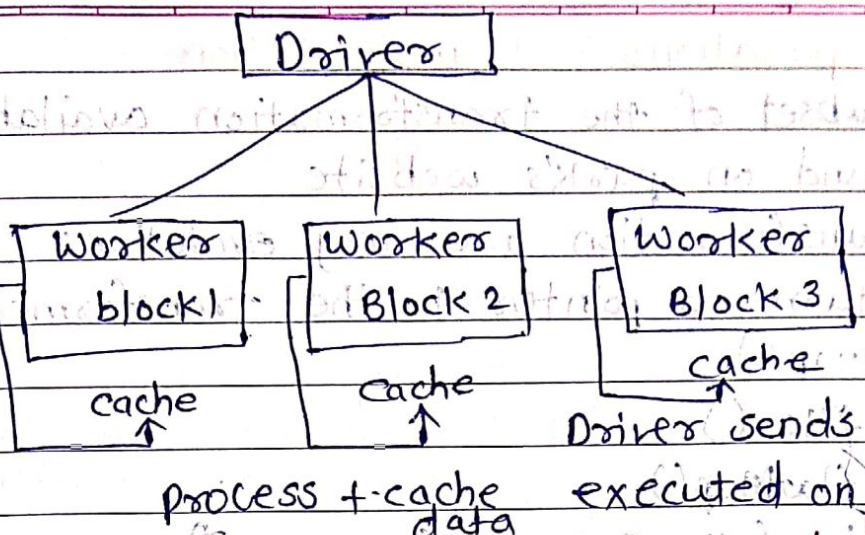
map reduce example:-

```
val wordcounts = textfiles.flatmap(line => line.split(" ")
.map(word => (word,1))
.reducebyKey((a,b) => a+b)

wordcounts.collect()
```

Direct Acyclic Graph (DAG)

- View the DAG
  linesLength.toDebugString



Driver sends the code to be executed on each block.

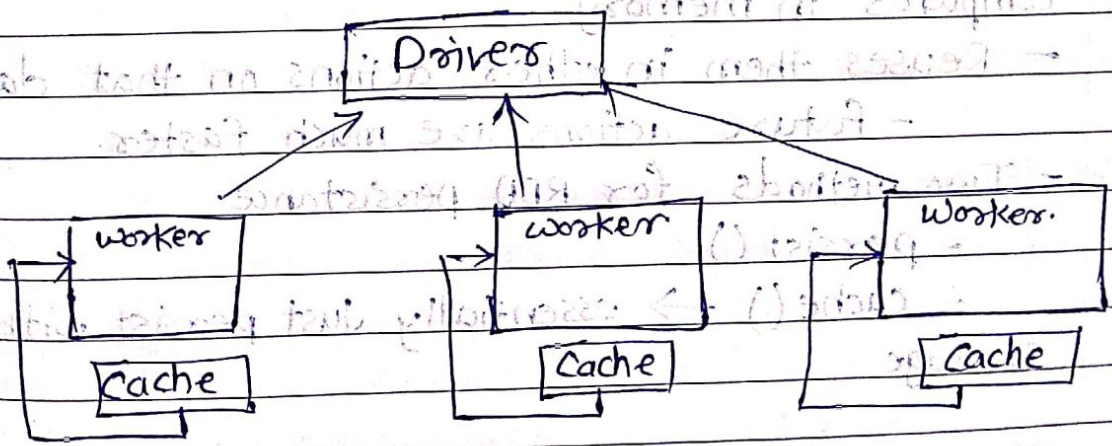
Process + cache data

What happens when an action is executed?

```
// creating the RDD
val logfile = sc.textFile("hdfs://...")

// Transformation.
val errors = logfile.filter(_.startsWith("Error"))
val messages = errors.map(_.split("\t")).map(r => r(1))

// caching
messages.cache()
// Actions
messages.filter(_.contains("mysql")).count()
messages.filter(_.contains("php")).count()
```



Process from cache

M T W T F S S
Page No.:
Date:
YOUVA

* RDD operations - Transformation
- A subset of the transformation available. full set can be found on spark's website
- Transformation are lazy evalutions.
- Returns a pointer to the transformed RDD
  1. map()
  2. filter()
  3. flatMap()
  4. Join(otherDataset, [Num Tasks])
  5. reduceByKey(func)
  6. sortByKey([ascending], [numTasks])

Action return values
  7. collect()
  8. Count()
  9. first()
  10. take(n)
  11. foreach(func)

* RDD - persistence
- Each node stories any partitions of the cache that it computes in memory
- Reuses them in other actions on that dataset
  - future actions are much faster.
- Two methods for RDD persistance
  - persist()
  - cache() → essentially just persist with memory_only storage

Storage level :-
Memory_only

Memory_AND DISK

MEMORY_ONLY_SER

MEMORY_AND_DISK_SER

DISK_ONLY

MEMORY_ONLY_2

MEMORY_AND_DISK_2, etc.

OFF_HEAP

* shared variables and key-value pairs
- when a function is passed from the driver to a worker, normally a separate copy of the variables are used.

Two types of variables:-
- Broadcast variables
  - read only copy on each machine
  - Distribute broadcast variables using efficient broadcast algorithms.
- Accumulators
  - Variables added through an associative operation
  - Implement counters and sums
  - Only the driver can read the accumulator value
  - Numeric types accumulators. Extend for new types

Scala: key-value pairs

```
val pair = ('a', 'b')
pair._1 // will return 'a'
pair._2 // will return 'b'
```

Python

```
pair = ('a', 'b')
pair[0] - # return 'a'
pair[1] - # return 'b'
```

M T W T F S S
Page No.:
Date:
YOUVA

* Programming with key value pair
  - There are special operations available the RDD's of key-value pairs
    - Grouping or aggregating elements by a key

- Tuple2 objects created by writing (a,b)
  - must import org.apache.spark.SparkContext._

- Pair RDDFunctions contains key-value pair operations
  - reduceByKey((a,b) ⇒ a+b)

- Custom objects as key in key-value pair requires a custom equals() method with a maching hashcode() method.

- Example :-

```
val textFile = sc.textFile("...")
val readmeCount = textFile.flatMap(line ⇒ line.split(" ")).map(word ⇒ (word, 1)).reduceByKey(_+_)
```

Module 3 : Spark Application Program

ming

Part 1 :- SparkContext sc

- The main entry point for spark functionality
- Represents the connection to a spark cluster
- create RDDs, accumulators and broadcast variables on that cluster
- In the spark shell, the SparkContext, sc is automatically initialized for you to use

In a Spark program, import some classes and implicit conversions into your program

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
```

Linking with spark - Scala

- Spark application requires certain dependencies
- Must have a compatible Scala version to write applications
  - e.g Spark 1.1.1 uses Scala 2.10

- To write a spark application, you need to add a Maven dependency on spark.
  - spark is available through maven central at:

```
groupId = org.apache.spark
artifactId = spark-core_2.10
version = 1.1.1
```

- To access a HDFS cluster, you need to add a dependency on hadoop-client for your version of HDFS.

```
groupId = org.apache.hadoop
artifactId = hadoop-client
version = <your-hdfs-version>
```

Linking with spark - Python

- Spark 1.1.1 works with python 2.6 or higher.
- Uses the standard CPython interpreter, so C libraries like Numpy can be used.
- To run spark application in python, use the bin/spark-submit script located in the spark directory.
  - load Spark's Java/scala libraries.
  - Allow you to submit applications to a cluster.

- If you wish to access HDFS, you need to use a build of Pyspark linking to your version of HDFS
- Import some Spark classes

```
from pyspark import SparkContext, SparkConf
```

* Initializing Spark - Scala

- Build a SparkConf object that contains information about your application

```
val conf = new SparkConf().setAppName(appName).
           setMaster(master)
```

- The appName parameter → Name for your application to show on the cluster UI
- The master parameter → is a spark, Mesos, or YARN cluster URL (or a special "local" string to run in local mode)
  - In testing, you can pass "local" to run Spark
  - local[16] will allocate 16 cores
  - In production mode, do not hard code master in the program. Launch with spark-submit and provide it there
- Then, you will need to create the SparkContext object

```
new SparkContext(conf)
```

```
conf = SparkConf().setAppName(appName).setMaster
       (master)

sc = SparkContext(conf=conf)
```

Passing functions to spark

- Spark's API relies on heavily passing functions in the driver program to run on the cluster three methods
- Three methods
  - Anonymous function syntax

```
(x: Int) ⇒ x + 1
```

- Static methods in a global singleton object

```
object myfunctions {
    def func1 (s: string) : String = {...}
}
myRdd.map (myfunctions.func1)
```

- Passing by reference
  - To avoid sending the entire, consider copying the function to a local variable
  e.g- val field = "Hello"

Consider:

```
def doStuff(rdd : RDD[string]) : RDD[string] = {
  val field_ = this.field
  rdd.map(x ⇒ field_ + x)}
```

* Programming the business logic
- Spark's API available in scala, Java or python.
- Create the RDD from an external dataset or from an existing RDD
- Transformation and actions to process the data
- Use RDD persistence to improve performance
- Use broad cast variables or accumulators for specific use cases.

* Create Spark standalone applications - Python

```
from pyspark import SparkContext ← import statement

logfile = "Your_spark
→ sc = SparkContext ("local", "simple App")
rkContext log data = sc.textfile (logfile).cache()
```

numAs = logData.filter(lambda s: 'a' in s).count()
→ numBs = logData.filter(lambda s: 'b' in s).count()

Transformations
+
Actions

* Run standalone applications
- Define the dependencies - can use any system builds (Ant, sbt, maven)
- scala → simple.sbt
- Python → --py-files argument

- Create the typical directory structure with the files Scala using SBT:
./simple.sbt
./src
./src/main
./src/main/scala
./src/main/scala/simpleApp.scala

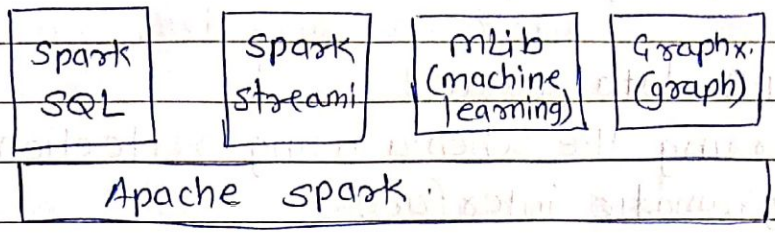
- Create a jar package containing the application's code
- Scala: sbt
- Python: submit-spark

- Use spark-submit to run the program

Module 4: Introduction to the spark libraries

spark libraries part-1
- Extension of the core spark API
- Improvements made to the core are passed to these libraries

- Little overhead to use with the spark core



Spark SQL

- Allows relational queries expressed in
  - SQL
  - HiveQL
  - Scala.
- SchemaRDD
  - Row objects
  - Schema
  - Created form
    - Existing RDD
    - parquet file
    - JSON dataset
    - HiveQL against Apache Hive
- supports scala, Java and python.

Spark SQL - Getting started

- SQLContext
  - Created from a sparkContext

Scala

```
val sc : SparkContext
val sqlcontext = new org.apache.spark.sql.SQLCon-
text (sc)
```

Python :-

```
from pyspark.sql import SQLContext sqlContext
                        =SQLContext (sc)
```

- import a library to convert an RDD to a SchemaRDD

- Scala only: import sqlContext.createSchemaRDD

-SchemaRDD data sources
- inferring the schema using reflection
- programatic interface

spark SQL - Infering the schema using reflection
- The case class in scala defines the schema of the table

```
case class person (name : string , age : Int)
```

- The arguments of the case class becomes the names of the columns.
- Create the RDD of the Person object

```
val people =
sc.textFile ("example/src/main/people.text")
.map (_.split (","))
.map (p => Person(p(0), p(1).trim.toInt))
```

- Register the RDD as a table

```
people.registerTempTable ("people")
```

- Run SQL statements using the sql method provided by the SQL Context

```
Val teenagers = sqlContext.sql ("select name from people where age >=13 and age<= 19")
```

-The results of the queries are Schema RDD. Normal RDD operations also work on them

```
teenagers.map (t => "Name : " + t(0)).collect().foreach (println).
```

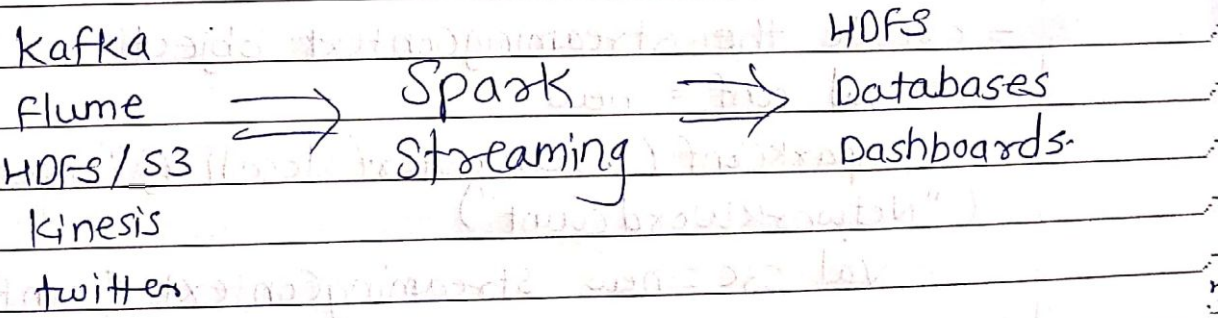
Spark Streaming

- Scalable, high-throughput, fault tolerant stream processing of live data streams.

- Receives live input data and divides into small batches which are processed and returned as batches
- Dstream - sequence of RDD
- Currently supports scala and Java.

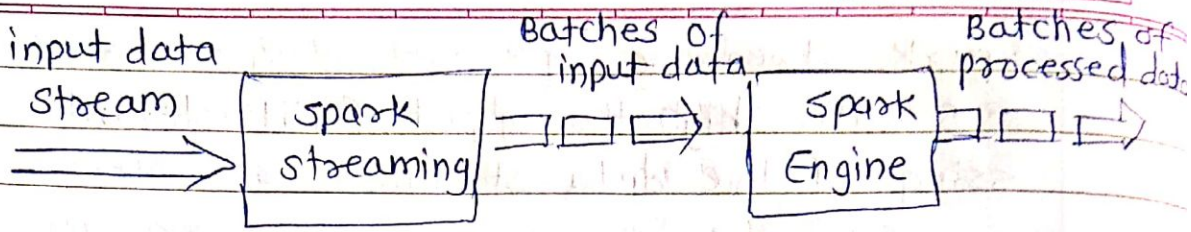
- Receives data from
  - Kafka
  - Flume
  - HDFS/S3
  - Kinesis
  - Twitter

- Pushes data out to:
  - HDFS
  - Databases
  - Dashboard.

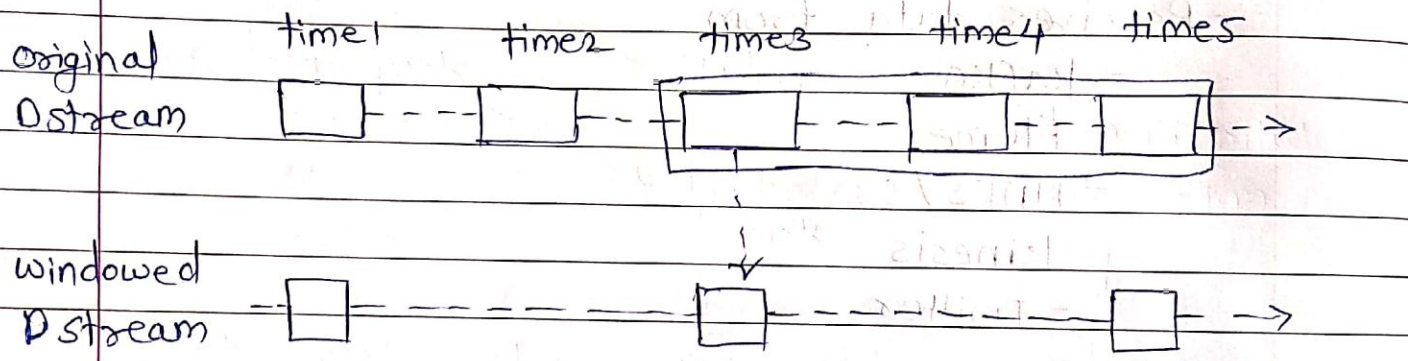


Spark Streaming - Internals:

- The input stream (Dstream) goes into spark streaming
- Breaks up into batches
- feeds into the spark engine for processing
- Generate the final results in stream of batches



- sliding window operations
  - windowed computations
    - window length
    - sliding interval
    - reduceByKeyAndWindow



spark streaming - Getting started

Scenario: count the number of words coming in from the TCP socket

- import the spark streaming classes & some implicit conversions
- create the StreamingContext object

```
val conf = new
  SparkConf().setMaster("local[2]").setAppName
("NetworkWordcount")
  val ssc = new StreamingContext (conf, seconds(1))
```

- Create a DStream

```
val lines = ssc.socketTextStream ("localhost", 9999)
```

- split the lines into words

```
val words = lines.flatMap(_.split(" "))
```

counts the words

```
val pairs = words.map(word => (word, 1))
val wordcounts = pairs.reduceByKey(_ + _)
```

Print to the console:

```
wordcounts.print()
```

Simplilearn Course :-

Course 1 :- Apache Spark Beginners Course

Introduction Apache Spark

Open-source Cluster Computing framework

- MapReduce
- Real time Processing
- Larger data on the network

- Basics of Big data and Apache spark
- Architecture of Apache Spark
- Install Apache spark on windows and ubuntu

What's in it for you?

- All about ~~you~~ apache spark
- Hadoop vs spark
- Components of Apache sparks
- Spark Architecture
- Application of spark
- Running a spark application
- Apache Spark Installation on windows
- Apache spark Installation on Ubuntu
- spark streaming

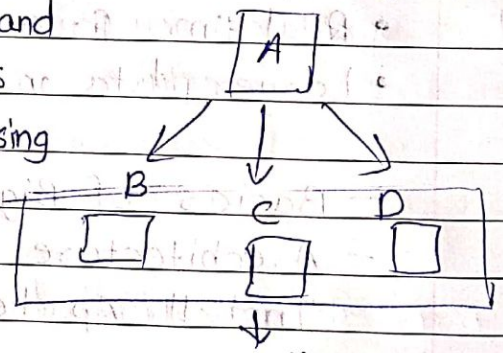
- Demo on spark streaming
- Spark MLib
- Spark SQL
- Pyspark
- Spark MLib Demo
- Spark SQL Demo
- Pyspark Demo
- Spark interview questions

1 All about Apache spark :-

- 5V's of Big data.

Big data spark → Cassandra
Big data spark → Hadoop → Hadoop Distributed file system.
Big data spark → Apache spark

Hadoop Distributed file system.
↓
Hadoop mapreduce



* Apache spark :- It is an open source data processing engine to store and process data in real-time across various clusters of computers using simple programming constructs

* Support languages
  - R
  - Scala
  - python
  - Java

* Developers and data scientists incorporate spark into their applications to rapidly query, analyze, and transform data at scale :- Query, Analyze, Transform

2 Hadoop vs spark

Hadoop — Processing data using MapReduce in Hadoop is slow

M T W T F S S

Page No.: YOUVA

Date:

spark :- spark processes data 100 times faster than mapReduce as it is done-in-memory.

	Hadoop	spark
2	Perform batch processing of data	2. Performs both batch processing and real-time processing of data
3	Hadoop has more lines of code. Since it is written in Java, it takes more time to execute	3. Spark well has fewer lines of code as it is implemented in scala, python.
4	Hadoop supports kerberos authentication which is difficult to manage	4. spark supports authentication via a shared secret. It can also run on Yarn leveraging the capability of kerberos.

Spark features :-

1) Fast processing

It contains RDD which saves time taken in reading, and writing operations and hence, it runs almost 10 to 1000 times faster than Hadoop.

2) In memory computing

In spark, data is stored in the RAM, so it can access the data quickly and accelerate the speed of analytics

3) Flexible

- Supports multiple language

4) fault tolerance

It contains RDD that are designed to handle the

failure of any worker node in the cluster. Thus, it ensures that the loss of data reduces to zero

5) Better analytics:-

spark has a rich set of SQL queries, machine learning algo, complex analytics etc. with all these functionalities analytics can be performed better.

3 Component of Apache spark.

- spark core
- spark SQL - structured, dataframe
- spark streaming
- MLlib - ML algo, prescriptive, descriptive
- GraphX

1) Spark core.

Spark core is the base engine for large-scale parallel and distributed data processing.

It is responsible for:

- memory management
- fault recovery
- scheduling, distributing and monitoring jobs on a cluster
- Interacting with storage system.

Resilient Distributed Dataset

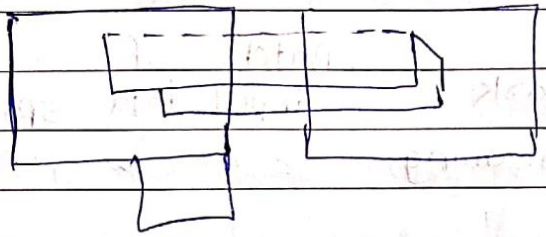
-spark core is embeded with RDDs (Resilient distributed datasets) an immutable fault-tolerant, distributed collection of objects that can be operated on in parallel

RDD

Transformation Action.

-These are operations (such as map, filter, join, union) that are performed on an RDD that yields a new RDD containing the result

These are operations (such as reduce, first, count) that return a value after running a computation on an RDD



Trigger drag ↓

- RDD1 — Base RDD parent RDD
- RDD2 — child RDD
- RDD3 —

method.

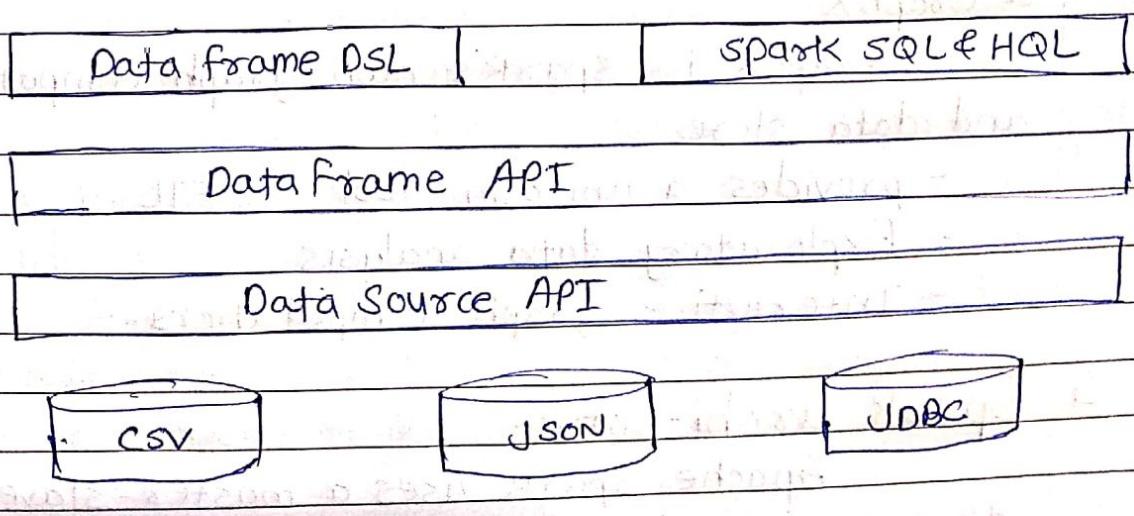
```
val x = sc.textfile
val y = x.map
val z = y.filter()
```

z.cout → action

Transformation

2. Spark SQL

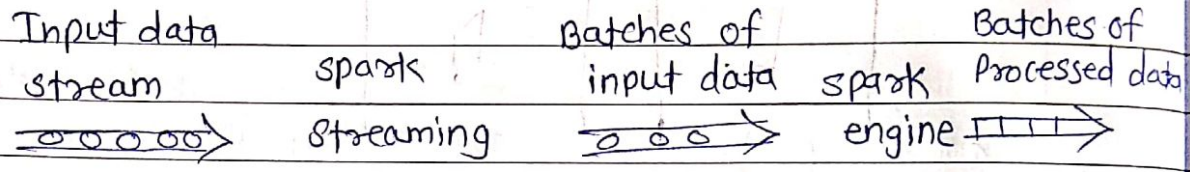
Spark SQL framework component is used for structured and semi-structured data processing



3. Spark Streaming

Spark Streaming is a lightweight API that allows developers to perform batch processing and real-time streaming of data with ease.

Provides secure, reliable and fast processing of live data streams



4. Spark MLib

MLib is a low level machine learning library that is simple to use is scalable and compatible with various programming languages.

MLib eases the deployment and development of scalable machine learning algorithms

It contains machine learning libraries that have an implementation of various machine algorithms

- clustering
- classification
- Colaborative filtering

5. GraphX

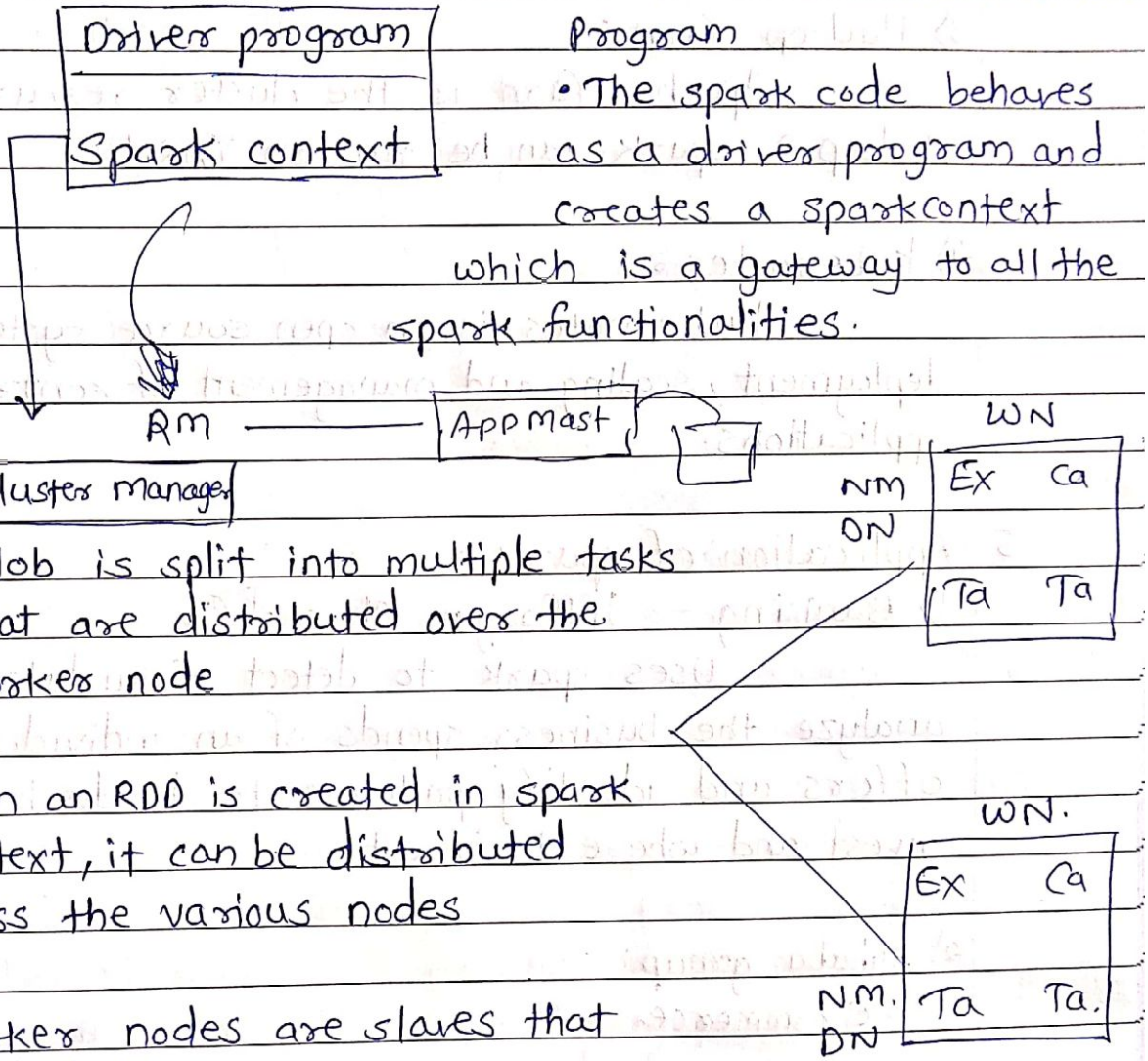
GraphX is Spark's own graph computation engine and data store

- provides a uniform tool for ETL
- Exploratory data analysis
- Interactive graph computations.

4 Spark architecture

Apache Spark uses a master-slave architecture that consist of a driver, that runs on a master

node, and multiple executors which run across the worker nodes in the cluster



- Master Node has a Drivers Program
- The spark code behaves as a driver program and creates a sparkcontext which is a gateway to all the spark functionalities.

- A Job is split into multiple tasks that are distributed over the worker node

- When an RDD is created in spark context, it can be distributed across the various nodes

- Worker nodes are slaves that run different tasks

Container = combination of RAM & CPU core

Spark cluster mangers.

1) Spark ~~clon~~ standalone mode

By default, application submitted to the standalone mode cluster will run in FIFO order and each application will try to use all available nodes.

2) Mesos :-

Apache mesos is an open source project to manage computer clusters, and can also run Hadoop applications

3) Hadoop Yarn :

Apache Yarn is the cluster resource manager of Hadoop 2 spark can be run on YARN.

4) Kubernetes :-

Kubernetes is an open source system for automating deployment, scaling and management of containerized applications.

5 Application of spark :-

1) Banking – JPMorgan Chase & Co.

Uses spark to detect fraudulent transactions, analyze the business spends of an individual to suggest offers and identify patterns to decide how much to invest and where to invest

2) Alibaba group :

e commerce

Use to analyze large sets of data such as real time transaction details, browsing history etc. in the form of spark jobs and provides recommendations to its users.

3) IQVIA (Healthcare)

IQVIA is a leading healthcare company that uses spark to analyze patients data, identify possible health issues and diagnose it based on their medical history

4) Entertainment - Netflix / RIOT games.

Entertainment and gaming companies like netflix and Riot games use to showcase relevent advertisements to their users based on the videos that they watch, share and like

Use case

Conviva

One of the world's leading video streaming companies

6 Running a spark application

IDE - Scala code - import packages.

How to spark application runs on a cluster?

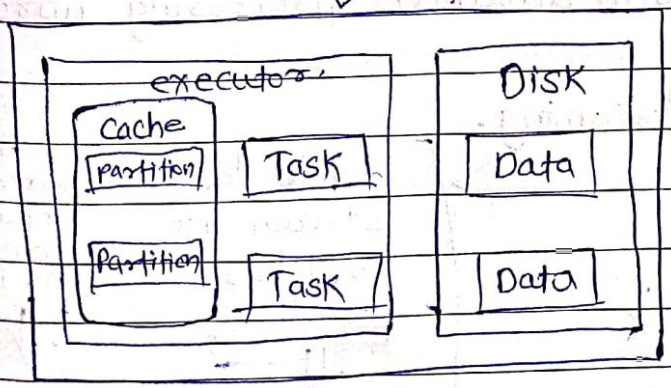
Driver program
Application

Spark Session

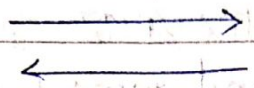
⇒ Spark applications run as independent processes, coordinated by the Spark Session object in the driver program

Resource manager
Cluster manager.

The resource or cluster manager assigns tasks to workers, one task per partition.



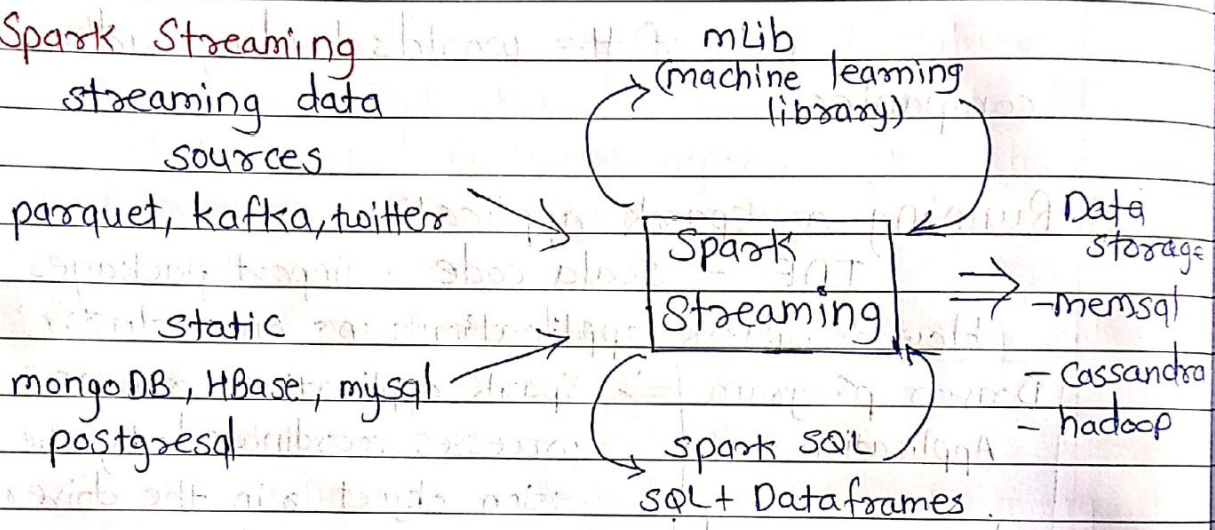
- A task applies its unit of work to the dataset in its partition and outputs a new partition dataset
- Because iterative algorithm apply operations repeatedly to data, they benefit from caching



results are sent back to the driver application or can be saved to disk

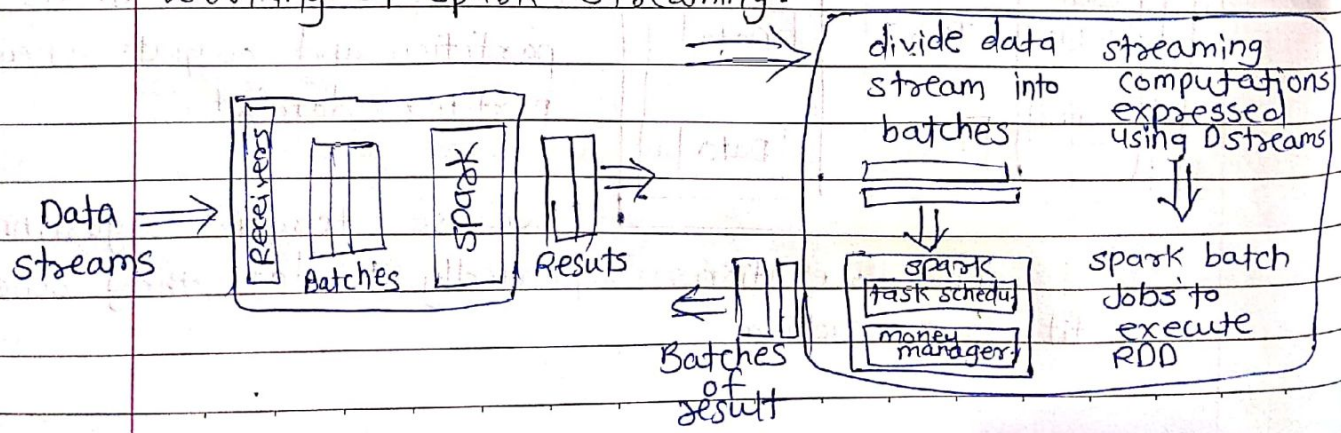
7 Apache spark installation on windows
8 Apache spark installation on ubuntu

9 Spark Streaming



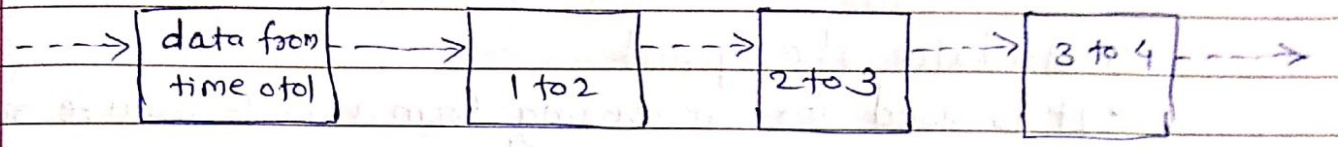
Features of spark streaming
- Fast recovery from failures
- better load balancing & resource usage
- Combining the streaming data with static datasets and interactive queries
- Native integration with advanced processing libraries.

Working of spark streaming.



Discretized Streams (Dstreams)

Discretized stream is the basic abstraction provided by spark streaming. It represents a continuous stream of data either the input data stream received from the source or the processed data stream generated by transforming the input stream



Windowed Stream processing

spark streaming allows you to apply transformation over a sliding window of data. This operation is called as windowed computation

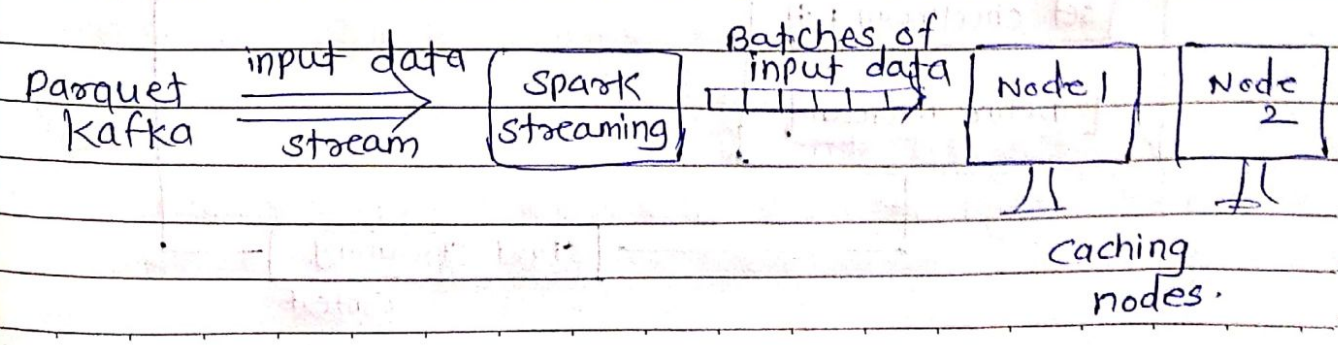
Fig see at back.

Caching/Persistence

DStreams allows developers to persist the stream data in memory.

With the help of persist() method, DStream will automatically persist every RDD of that DStream in memory.

The default persistance level is set to replicate the data to two nodes for fault tolerance, for input streams that receives data over the network



Checkpointing in spark Streaming

Checkpointing is the process to make streaming applications resilient to failures

To ensure streaming applications operate in a 24x7 environment, you should checkpoint enough information for a storage system that is fault-tolerant

Metadata checkpoint

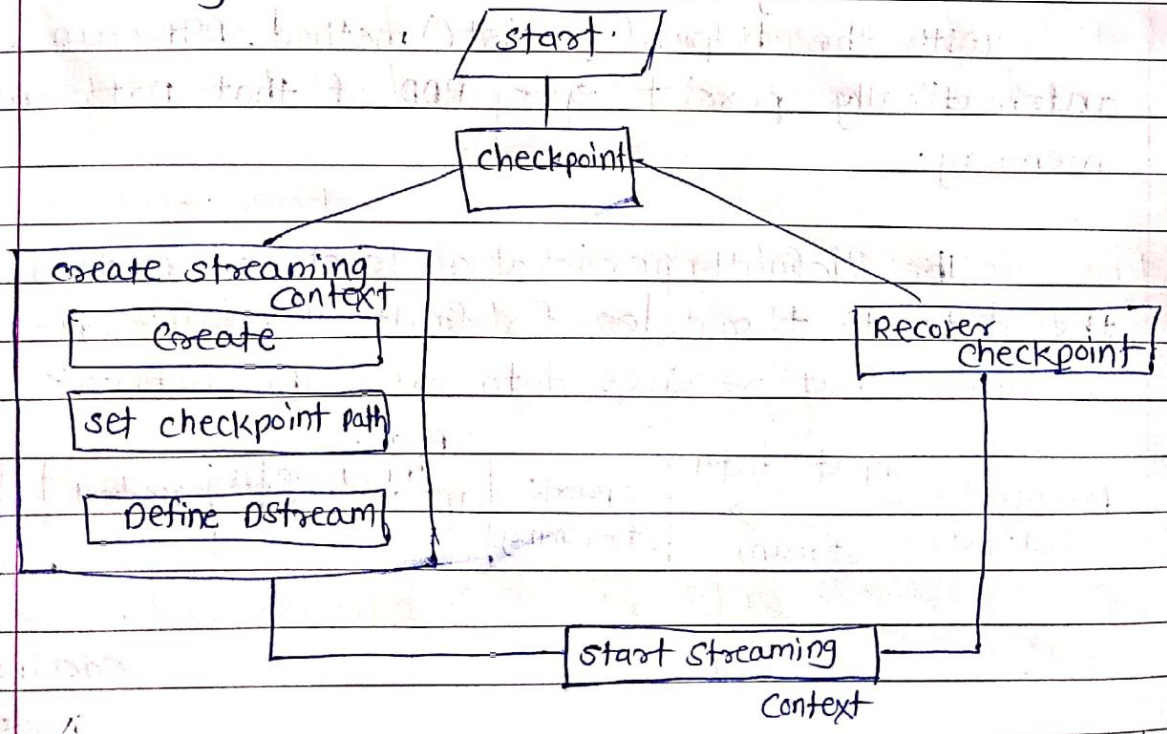
- It is used for recovering from a node failure running this streaming application driver

It has metadata that comprises of

- Streaming application configuration
- Incomplete batches
- DStream operations defining the streaming application

Data checkpoint

- It saves the information defining the streaming computation to fault-tolerant storage like HDFS
- It is used in a few stateful transformations combining data across various batches



Accumulators in spark streaming

Accumulator are variable that are only added through an associative and commutative operation.

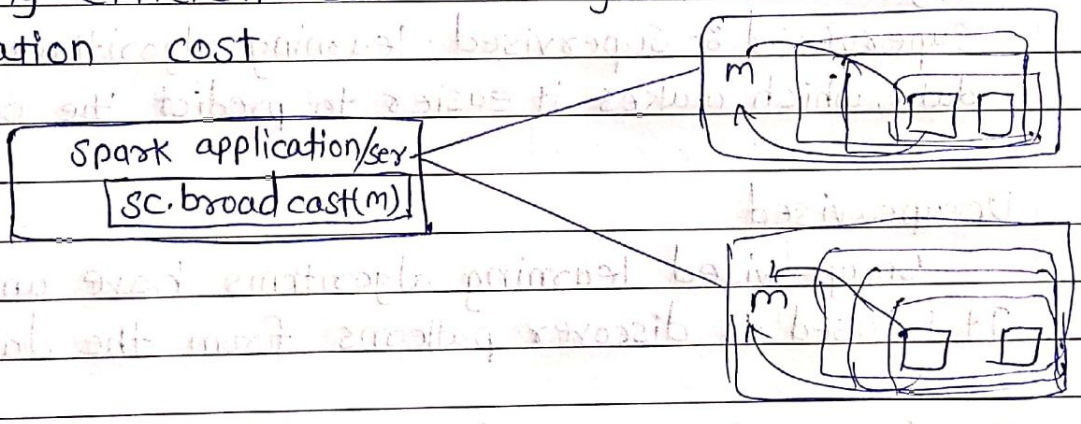
Spark natively supports accumulators of numeric types and programmers can add support for new types.

Tracking accumulators can be useful for understanding the progress of running stages.

Broadcast variables in spark streaming

Broadcast variables allow the programmer to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks.

spark attempts to distribute broadcast variables using efficient broadcast algorithms to reduce communication cost



Spark streaming Example:-

spark streaming is widely used in retail chain companies.

10 Demo on spark streaming

11 Spark MLlib

MLlib stands for machine learning library in spark. The goal of this library is to make practical machine learning scalable and easy to implement.

Page No.:
Date:
YOUVA

Spark MLlib ⟹ Collects data ⟹ Processes data ⟹ Builds data model

What is machine learning?

Machine learning is the science of training computers to learn from data and perform specific tasks without being explicity programmed

Past data ⟹ [learn from past data] ⟹ Predicts the output

Machine learning Algorithms can be classified into 3 main categories

Supervised :- Supervised learning algorithms have labled data, which makes it easier to predict the output

Unsupervised

Unsupervised learning algorithms have unlabled data It is used to discover patterns from the data.

Reinforcement

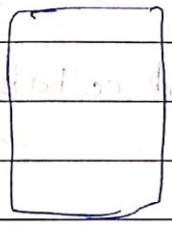
In reinforcement learning algorithms, an agent, receives a reward in the next time step to evaluate it's previous action.

Supervised Learning :

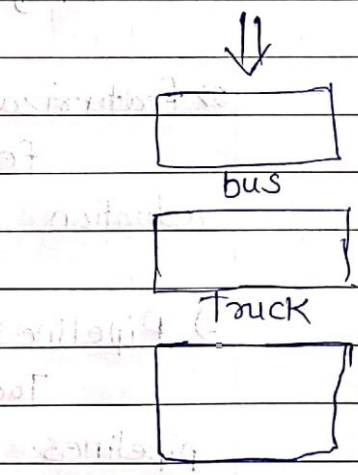
Labelled data of dogs and cats ⟹ Data fed to the algo ⟹ Data Processing ⟹ Trained model (input ↓) ↓ This is dog

supervised learning is classified into Regression and Classification
- Linear Regression
- Logistic Regression
- Decision Tree
- Random forest
- Support Vector Machines
- K Nearest Neighbors
- Naive Bayes

Unsupervised learning



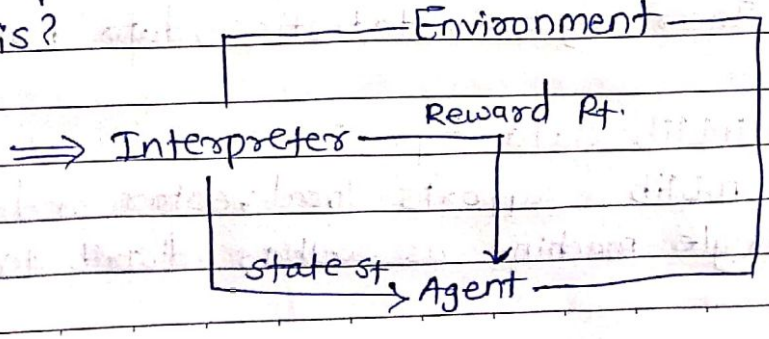
⟹ Data fed to the algorithm ⟹ Learns the patterns in the data ⇒ Trained model

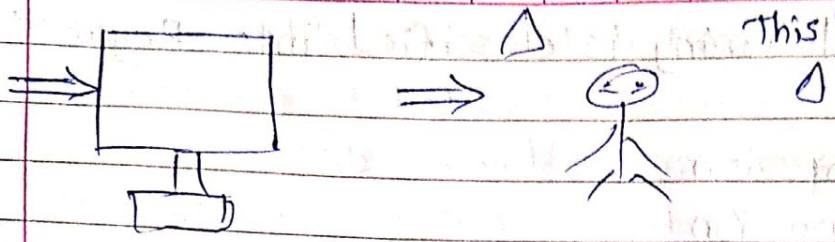


Unsupervised learning is classified into Clustering and Association rules:
- K means clustering
- Hierarchical Clustering
- DBSCAN
- Principal Component Analysis
- Gaussian Mixture model
- Hidden markov Process

Reinforcement Learning

what shape is this?





Below are the popular algorithm in Reinforcement learning
- Q learning
- Temporal learning
- Generative adversarial Networks

Spark MLib Tools

Spark's MLlib component provides the following tools:

1) ML Algorithms:

Classification, regression, clustering and collaborative filtering

2) Featurization

feature extraction, transformation, dimensionality reduction & selection.

3) Pipelines

Tools for constructing, evaluting and tunning ML pipelines.

4) Persistance

saving and loading algorithms models and pipelines

5) Utilities

Inner algebra, statistics, data handling

*Spark MLlib Data types:

MLlib supports local vector and matrices stored on a single machine as well as distributed matrices

M T W T F S S
Page No.:
Date:
YOUVA

Local vector:-

mllib supports two types of local vectors:
dense and sparse

Example :- vector (1.0, 0.0, 3.0)

dense format : [1.0, 0.0, 3.0]

Sparse format : (3, [0,2], [1.0, 3.0])

* Labeled point (supervise learning algorithm).

A labeled point is a local vector, either dense or sparse that is associated with a label/response

e.g. In binary classification, a label should be either 0 (negative) or 1 (positive)

* Local matrix

A local matrix has integer type row and column indices, and double type values that is stored in a single machine

$$\begin{bmatrix} 1.0 & 2.0 \\ 3.0 & 4.0 \\ 5.0 & 6.0 \end{bmatrix}$$

matrix (3×2)

[1.0, 3.0, 5.0, 2.0, 4.0, 6.0]

1-D Array.

* Distributed matrix

A distributed matrix has long-type row and column indices and double-type values. It is stored in a distributed manner in one or more RDDs

Types of distributed matrix

- Row matrix
- Indexed RowMatrix
- Coordinate matrix

* Spark MLlib Basic Statistics

1> Summary Statistics :-

MLlib provides summary statistics for RDD (vector) through the function colStats() available in Statistics package.

colstats() returns the column-wise max, min, mean, variance, number of nonzero values, and the total count.

2) Correlations

MLlib can perform correlation between two series of data using the statistics package.

Based on the type of input, if there are two RDDs [double] or an RDD [vector] the output will be a Double or the correlation matrix

3) Stratified Sampling

Stratified sampling methods, sampleByKey and sampleByKeyExact, can be performed on RDD's of key value pairs

sampleByKeyExact() is currently not supported in python

The sampleByKey()
will flip a coin to decide wheather an observation will be sampled or not

4) Random data generation

Random data generation is used for randomized algorithms, prototyping, and performance testing.

RandomRDDs package provides factory methods to generate random double RDDs or vector RDDs

Code to generate a random double RDD, whose values follows the standard normal distribution N(0,1) and then map it to N(1,4):